01-软件设计到底是什么？

你好！我是郑晔。

一个软件需要设计，这是你一定认同的。但软件设计到底是什么，不同的人却有着不同的理解：

- 有人认为，设计就是讨论要用什么技术实现功能；
- 有人认为，设计就是要考虑选择哪些框架和中间件；
- 有人认为，设计就是设计模式；
- 有人认为，设计就是Controller、Service加Model；
- ……

你会发现，如果我们按照这些方式去了解“软件设计”，不仅软件设计的知识会很零散，而且你会像站在流沙之上一般：

- 今天你刚学会用Java，明天JavaScript成了新宠，还没等你下定决心转向，Rust又成了一批大公司吹捧的目标；
- 你终于知道了消息队列在解决什么问题，准备学习强大的Kafka，这时候有人告诉你Pulsar在某些地方表现得更好；
- 你总算理解了Observer模式，却有人告诉你JDK中早就提供了原生的支持，但更好的做法应该是用Guava的EventBus；
- 你好不容易弄清楚MVC是怎样回事，却发现后端开发现在的主要工作是写RESTful服务，Controller还没有用，就应该改名成Resource了；
- ……

我们说，软件设计要关注长期变化，需要应对需求规模的膨胀。这些在不断流变的东西可能还没你的软件生命周期长，又怎能支撑起长期的变化呢！

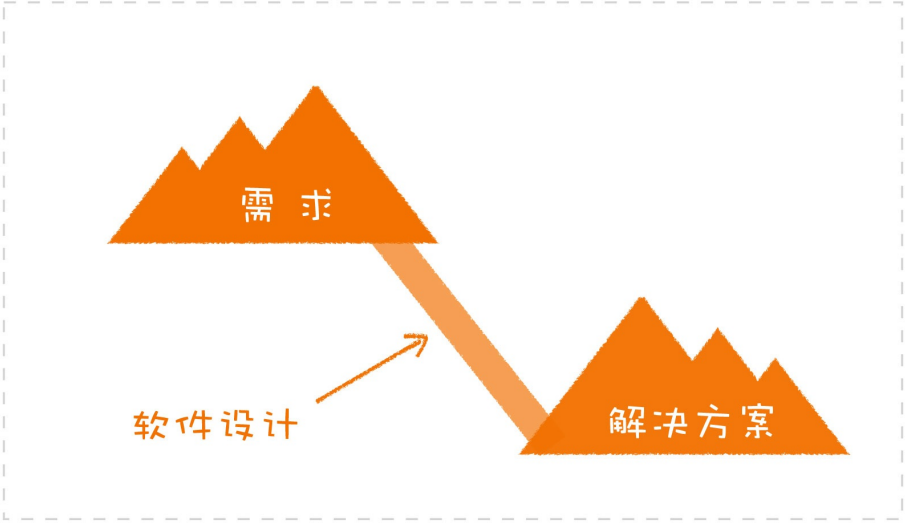
那么回到一开始的问题，软件设计到底是什么呢？

核心的模型

在回答这个问题之前，我们先来思考这样一件事：软件的开发目的是什么？

一个直白的答案就是，软件开发是为了解决由需求带来的各种问题，而解决的结果是一个可以运行的交付物。比如，我们在线购物的需求，是通过电商平台这个方案解决的。

那软件设计在这个过程中做的事情是什么呢？就是在需求和解决方案之间架设一个桥梁。



区别于解决简单的问题，软件的开发往往是一项长期的工作，会有许多人参与其中。在这种情况下，就需要建立起一个统一的结构，以便于所有人都能有一个共同的理解。这就如同建筑中的图纸，懂建筑的人看了之后，就会产生一个统一的认识。

而在软件的开发过程中，这种统一的结构就是模型，而**软件设计就是要构建出一套模型**。

这里所说的模型，不仅包括用来描述业务的各种实体，也包括完成业务功能的各种组件。人们写代码中常常会用到的服务（Service）、调度器（Scheduler）等概念就是一个一个的模型。

模型，是一个软件的骨架，是一个软件之所以是这个软件的核心。一个电商平台，它不用关系型数据库，还可以用NoSQL，但如果没有产品信息，没有订单，它就不再是电商平台了。

可能有不少人一听到模型，就会情不自禁地要打退堂鼓，认为这些内容过于高大上，其实大可不必，**模型的粒度可大可小**。如果把模型理解为一个一个的类，是不是你就会熟悉很多了，这就是小的模型。你也可以把一整个系统当作一个整体来理解，这就是大的模型。

关于设计，你一定听说过一个说法，“**高内聚、低耦合**”，**这其实就是对模型的要求。**一个“高内聚、低耦合”的模型能够有效地隐藏细节，让人理解起来也更容易，甚至还可以在上面继续扩展。比如，我们后面课程会讲到的程序设计语言，就是提供了一个又一个的编程模型，让我们今天写程序不用再面对各种硬件的差异，还能够在此基础上继续提供新功能。

你在日常工作中用到的各种框架和技术，也是提供了一个又一个的模型，它们大幅度降低了我们的开发门槛。所以你看，整个计算机世界就是在这样一个又一个模型的叠加中，一点一点构建出来的。用一个程序员所熟悉的说法就是：模型是分层的。这就像乐高一样，由一个个小块构建出一个个大一些的部件，再用这些部件组成最终的成品。

这与一些人常规理解的Controller、Service那种分层略有差异。但实际上，这才是在计算机行业中普遍存在的分层。我们熟悉的网络模型就是一个典型的分层模型。按照TCP/IP的分层方法，网络层要构建在网络接口层之上，应用层则要依赖传输层，而我们平时使用的大多数协议则属于应用层。

即便是在一个软件内部，模型也可以是分层的。**我们可以先从最核心的模型开始构建，有了这个核心模型之后，可以通过组合这些基础的模型，构建出上面一层的模型**。

我曾经做过一个交易系统的设计。在分析了主要的交易动作之后，我提出了一个交易原语的概念，包括资产冻结、解冻、出金、入金等少数几个动作。然后，把原先的交易动作变成了原语的组合。比如，下单是资产冻结，成交是不同账户的出金和入金，撤单则是资产解冻。

在这个结构下，由交易原语保证每个业务的准确性，由交易动作保证整个操作的事务性。从上面这个图中，你可以看出，这就是一种分层，一种模型上的分层。

好，到这里我们已经对软件设计中的模型有了一个初步的认识。总结一下就是，模型是一个软件的核心；模型的粒度可大可小；好的模型应该“高内聚、低耦合”；模型可以分层，由底层的模型提供接口，构建出上层的模型。

后续我们这个课程的大部分内容都会围绕着模型来讲：怎样理解模型、建立模型、评判模型的优劣等等。

学会这些知识之后，能在多大的粒度上应用它们，你就能掌控多大的模块。不过，仅仅是把软件设计理解成

构建模型，这个理解还不够。模型设计也不能任意妄为，需要有一定的约束，而这个约束，就是软件设计要构建的另一个部分：规范。

约束的规范

如果说，软件设计要构建出一套模型，这还是比较直观好理解的。因为模型通常可以直接体现在代码中。但软件设计的另一部分——规范，就常常会被忽略。

规范，就是限定了什么样的需求应该以怎样的方式去完成。比如：

- 与业务处理相关的代码，应该体现在领域模型中；
- 与网络连接相关的代码，应该写在网关里；
- 与外部系统集成的代码，需要有防腐层；
- ……

其实，每个项目都会有自己的规范。比如，你总会遇到一些项目里的老人，他们会告诉你，这个代码应该写在这，而不应该写在那，这就是某种意义上的规范。虽然规范通常都有，但问题常常也在。

一种常见的问题就是缺乏显式的、统一的规范。

规范的一个重要作用，就是维系软件长期的演化。如果没有显式的规范，项目的维系只能依靠团队成员个人的发挥，老成员一个没留神，新成员就可能创造出一种诡异的新写法，项目就朝着失控又迈出了一步。

不知道你是否接触过这样的项目，多种不同的做法并存其中：

- 数据库访问，有用MyBatis的，有用JDBC的，也有用Hibernate的；
- 外部接口设计，有用REST风格的，有用URL表示各种动作的；
- 文件组织，有的按照业务功能划分（比如，产品、订单等），有的按照代码结构划分（比如，Resource、Service等）；
- ……

没有一个统一的规范，每一个项目上的新成员都会痛斥一番前人的不负责任。然后，新的人准备另起炉灶，增加一些新东西。这种场景你是不是很熟悉呢？混乱通常就是这样开始的。

如果存在一个显式的、统一的规范，项目会按照一个统一的方向行进。即便未来设计要演化、规范要调整，有一个统一的规范也要比散弹打鸟来得可控得多。

关于规范，**还有一种常见问题就是，规范不符合软件设计原则**。我给你讲一个让我印象深刻的故事。

我曾经遇到一个网关出现了OOM（Out of Memory，内存溢出）。这个网关日常的内存消耗高达150G，一次流量暴增它就扛不住了。后来经过优化，把内存消耗降到了8G。

如果单看数字，这是一个接近20倍的优化，大手笔啊，但这里面究竟发生了什么呢？实际上，这次优化最核心的内容就是构建了一个防腐层，将请求过来的JSON转换成了普通的内存对象。而原来的做法是把JSON解析器解析出来的对象到处使用，因为这些对象上附加很多额外的信息，导致占用了大量的内存。

很遗憾，这不是大牛战天斗地的故事，只是因为旧的规范不符合软件设计原则而导致的错误：外部请求的对象需要在防腐层转换为内部对象。

模型与规范

有了模型，有了规范，**那模型与规范是什么关系呢？模型与规范，二者相辅相成**。一个项目最初建立起的模型，往往是要符合一定规范的，而规范的制定也有赖于模型。这就像讨论户型设计时，你可以按照各种方式组合不同的空间（模型），却不会把厨房与卫生间放在一起（规范）。

至此，我们已经知道了，软件设计既包含构建出一套模型，也包括制定出相应的规范。再回过头来看这节课开头的问题，你是不是对软件设计有了重新的认识呢？特定技术、框架和中间件，只是支撑我们模型的实现，而设计模式、Controller、Service、Model这些东西也只是一个特定的实现结果，是某些特定场景下的模型。

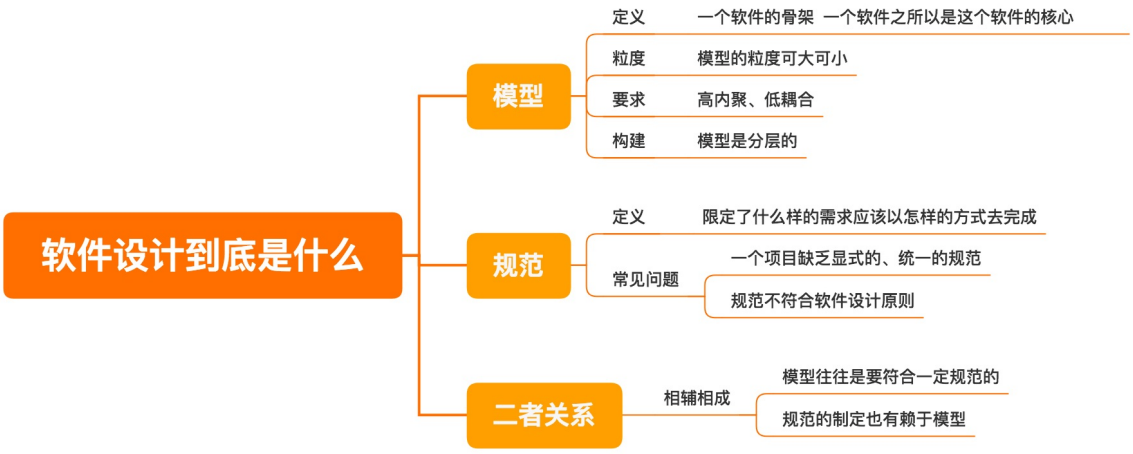
总结时刻

今天，我们学习了软件设计到底是什么，它应该包括“模型”和“规范”两部分：

- 模型，是一个软件的骨架，是一个软件之所以是这个软件的核心。模型的粒度可大可小。我们所说的“高内聚、低耦合”指的就是对模型的要求，一个好的模型可以有效地隐藏细节，让开发者易于理解。模型是分层的，可以不断地叠加，基于一个基础的模型去构建上一层的模型，计算机世界就是这样一点点构建出来的。
- 规范，就是限定了什么样的需求应该以怎样的方式去完成。它对于维系软件长期演化至关重要。关于规范，常见的两种问题是：一个项目缺乏显式的、统一的规范；规范不符合软件设计原则。
- 模型与规范，二者相辅相成，一个项目最初建立起的模型，往往是要符合一定规范的，而规范的制定也有赖于模型。

有了对软件设计的初步了解，我们就准备开始做设计了，但该从哪入手呢？这就是我们下一讲的内容。

如果今天的内容你只能记住一件事，那请记住：**软件设计，应该包括模型和规范**。



极客时间

思考题

最后，我想请你分享一下，你的项目是如何做设计的。欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- Kăfkă²º²º 2020-05-25 20:19:10
  业务讨论之后进行领域设计，画出出静态模型（包括子系统、模块等）和动态结构（交互等），或者先勾勒接口（内内外系统的区隔），再做模型。实际过程有很多反复，并且会进行角色代入，看模型能否支持业务，直到模型比较稳定 [2赞]

- 木云先森 2020-05-25 19:45:23
  还需要前面有个好的产品经理或是业务专家。以及公司有个好的文化。各种频繁的插队的需求，各种前后都无法闭环的需求。都是，软件产品异常大的阻碍 [2赞]

- 渔夫 2020-05-25 19:10:12
  很多软件产品的需求都是一点点冒出来的，甚至中途需求还会去溜出去绕个弯，然后又回归，设计有种被牵着鼻子走的感觉，工期紧迭代快，结果就是设计的模型中有大量名不符实的定义，还有很多定义的补丁，实在很糟心，当然需求发展方向终会明朗，这时候就需要重构整理，包括设计和实现，同时又要应对新的业务开发，于是形成了两线或多线作战，苦啊！这样的情况除了增加团队，不知道老师有什么好的建议？ [2赞]

- 小文同学 2020-05-25 23:32:39
  我独立设计的第一个项目整体来说，是失败的。就是盲目模仿前项目，没理解，分层，抽象，接口，模型等设计概念，最终项目陷入很麻烦的技术问题。 [1赞]

- 渺渺兮于怀 2020-05-25 23:14:02
  慢慢的，某个瞬间，突然觉得自己的工作不再是码农，而是软件设计，并且在工作中得到强烈的自我肯定。
  一个好的软件设计思路，首先是符合大众习惯行为、符合日常常理，其次再是数据模型设计、技术范畴设计。
  一个好的软件设计实现，往往可以很容易兼容正常合理的需求变更，对开发工作来说，掌握其核心，理论与实践相结合，可以事半功倍！ [1赞]

- 晴天了 2020-05-25 20:25:05
  设计了1程序 开发完成了。 这时候产品说 在1内部加一条小需求。。 这种情况是不是很抓狂 [1赞]

- 捞鱼的搬砖奇 2020-05-25 20:15:53
  熟悉的声音回来了。 [1赞]

- 业余爱好者 2020-05-26 09:24:52
  向贝佐斯学习，做事情要建立在不变的东西上。

  模型是一个理解世界的抽象模型，就像科学理论一样。好的模型应该是稳定的，简洁的。

  规范也不能朝令夕改，规范就是做事的高层原则，相当于“公理”。公理要么来自于根深蒂固的人性（广义的，中性的，如懒惰，两点之间直线最短），要么就是大量经验教训的积累（如业界各公司如阿里巴巴开发手册之类）。（这里看出为什么小公司规范意识稀缺了，踩得坑不足，积累经验不足，你就是想规范也规范不起来。解决办法就是参考业界或大公司的规范，当然不能照搬。）

- 段启超 2020-05-26 00:45:14
  防腐层是模型的一个规范，分享下我对防腐层的认知：

我接触防腐层的概念是从DDD的限界上下文开始的。Eric用细胞膜的概念来解释“限界”的概念，细胞膜只让细胞需要的物质进入细胞，同样，我们的代码之间业务也存在这个样一个界限，同一个对象的业务含义在不同的上下文中是不一样的。以在网上买书为例，在购买页面，我们的关注点在于这本书的名称，作者，以及分类，库存等信息；提交订单后，这本书就成为了订单上下文中的一个订单item，我们会关注这个item 的数量以及购买他的人是谁，以及书的配送地址等； 订单提交给仓库后，仓库会关心这本书还有没有库存，以及打包状态，分拣，物流等状态。
防腐层是在限界上下文之间映射（说白了就是交互）的方式，体现在代码上就是一个对象的转换，这个转换的意义在于隔离变化，防止因为对象在一个上下文中的变化扩散到其他的上下文中。

关于规范：
规范也是团队文化中很重要的一部分，以持续集成为例子，它的执行严格依赖于团队的开发纪律文化，以为了所谓赶进度而单元测试覆盖很低或者直接不写；采用分支策略方开发，一星期都合并不了主干，类似的人到处倒是，也就因为这一点，很多团队都在持续集成这个环节上掉队了。所以开发规范真的很重要，时刻谨记：混乱始于没有规范。

- Flynn 2020-05-26 00:12:46
  嗯。。项目的设计是视图、数据、模型

- Jxin 2020-05-25 23:12:59
  1.我现在在项目中采用ddd的分层架构。（不要求领域模型设计，仅限定了基本实现规范）
  2.因为整个公司缺乏显示统一的规范，我希望引入ddd的分层架构去限定这个规范。而且，项目本身虽是微服务技术栈，但模型本质还是大单体，用ddd挺好。（如果是真的微服务，不需要采用ddd的分层，ddd分层架构的理念应该在系统架构上去体现，落地到具体微服务包应该要对这些复杂性无感）
  3.起了分层的目录结构，做了下各层作用的讲解，前期大部分需求，我都定义好api，然后让队友们来实现。结果还是不理想。问题不在于知识难不难，而在个人意愿和市场需求。相对于个人代码质量的追求，大多数人更愿意把时间放在技术上。毕竟技术面试占分更高，且武学学会了就是会了，能做到分毫不差，短期既有价值。而内功除了学会，还要积累，短期难有成效。人往往都有点浮躁。

- zgscy100 2020-05-25 18:42:59
  方法论有了，如何落地是个问题

- 北天魔狼 2020-05-25 17:34:23
  老师好，我又来上课了